What is claimed is:

1. A filter compiler, comprising:

a filter coefficient generator arranged to provide a first set of filter coefficients corresponding to a filter spectral response;

a filter spectral response simulator coupled to the filter coefficient generator for providing an expected filter spectral response based in part upon the first set of filter coefficients;

a filter resource estimator coupled to the filter spectral response simulator for estimating an implementation cost of the filter; and

a filter compiler unit coupled to the filter resource estimator arranged to compile a filter implementation output file.

2. A compiler as recited in claim 1, wherein when the expected filter spectral response is substantially the same as the filter spectral response, then the first set of filter coefficients is a second set of filter coefficients.

3. A compiler as recited in claim 2, wherein the estimating the implementation cost of the filter is based upon the second set of filter coefficients

4. A compiler as recited in claim 3, wherein the filter is a finite impulse response (FIR) filter.

5. A compiler as recited in claim 4, wherein the filter coefficients are FIR filter coefficients, and wherein the filter spectral response is a FIR filter spectral response, and wherein the expected filter spectral response is an expected FIR filter spectral response, and wherein the desired filter implementation output file is a FIR filter implementation output file.

6. A compiler as recited in claim 5, wherein the desired FIR filter implementation output file comprises:

a FIR filter hardware implementation file; and

a FIR filter simulation file, wherein the FIR filter simulation file provides FIR filter simulation input data and wherein the FIR filter hardware implementation file provides a routing and placing dataset suitable for fitting the FIR filter in a programmable logic device.

7. A filter compiler as recited in claim 1, further comprising:

a floating-point filter coefficient converter coupled to the filter coefficient generator, wherein when the first set of filter coefficients is a set of floating-point filter coefficients, the floating-point filter coefficient converter converts the set of floating-point filter coefficients to a set of fixed-point filter coefficients.

8. A filter compiler as recited in claim 1, further comprising:

a filter input parameter buffer coupled to the filter coefficient generator arranged to store a plurality of filter input parameter values suitable for implementing the filter.

9. A filter compiler as recited in claim 8, wherein the plurality of filter input parameter values is a plurality of FIR filter input parameter values and wherein the filter is the FIR filter.

10. A filter compiler as recited in claim 9, wherein the plurality of FIR filter input parameter values include, a tap quantity value, a pipeline delay value, a parallel FIR filter architecture indicator value, a serial FIR filter architecture value, an input bandwidth value, and an initial set of FIR filter coefficient values.

11. A filter compiler as recited in claim 1, further comprising:

a filter rounding and scaling unit coupled to the filter coefficient generator, wherein when the initial set of filter coefficient values is a floating-point initial set of filter coefficient values, then the filter rounding and scaling unit scales and rounds the floating-point initial set of filter coefficient values to form the first set of filter coefficient values.

12. A filter compiler as recited in claim 5, wherein the a FIR filter simulation file is selected from a group comprising:

a MAX-PLUS2 vector file;

a MATLAB SIMULINK model;

a MATLAB TESTBENCH model;

a Verilog model; and

a VHDL model.

13. A filter compiler as recited in claim 5, further comprising:

a multi-rate FIR filter generator coupled to the input buffer arranged to build a multi-rate FIR filter selected from a group comprising: an interpolating FIR filter and a decimating FIR filter.

14. A filter compiler as recited in claim 1, wherein the filter is fitted in a programmable integrated circuit.

15. A filter compiler as recited in claim 14, wherein the programmable integrated circuit is a programmable logic device.

16. A method of compiling a filter having a selected filter spectral response, comprising:

providing a first set of filter coefficients corresponding to the filter spectral response;

providing an expected filter spectral response based in part upon the first set of filter coefficients;

comparing the desired filter spectral response to the expected filter spectral response;

estimating an implementation cost of the desired filter; and

compiling a filter implementation output file.

17. A compiler as recited in claim 16, wherein when the expected filter spectral response is substantially the same as the selected filter spectral response, then the first set of filter coefficients is a second set of filter coefficients.

18 A compiler as recited in claim 17, wherein the estimating the implementation cost of the filter is based upon the second set of filter coefficients

19 A compiler as recited in claim 18, wherein the filter is a finite impulse response (FIR) filter.

20. A compiler as recited in claim 16, wherein the filter coefficients are FIR filter coefficients, and wherein the selected filter spectral response is a FIR filter spectral response, and wherein the expected filter spectral response is an expected FIR filter spectral response, and wherein the desired filter implementation output file is a desired FIR filter implementation output file.

21. A method as recited in claim 20, wherein the FIR filter implementation output file comprises:

a FIR filter hardware implementation file; and

a FIR filter simulation file, wherein the desired FIR filter simulation file provides FIR filter simulation input data and wherein the desired FIR filter hardware implementation file provides a routing and placing dataset suitable for fitting the FIR filter in a programmable logic device.

22. A method as recited in claim 21, further comprising:

when the first set of FIR filter coefficients is a set of floating-point FIR filter coefficients,

converting the set of floating-point FIR filter coefficients to a set of fixed-point FIR filter coefficients.

23. A method as recited in claim 22, further comprising:

providing a plurality of FIR filter input parameter values suitable for implementing the desired FIR filter.

24. A method as recited in claim 23, wherein the plurality of FIR filter input parameter values include, a tap quantity value, a pipeline delay value, a parallel FIR filter architecture indicator value, a serial FIR filter architecture value, an input bandwidth value, and an initial set of FIR filter coefficient values.

25. A method as recited in claim 24, further comprising:

wherein when the initial set of FIR filter coefficient values is a floating-point initial set of FIR filter coefficient values,

scaling and rounding the floating-point initial set of FIR filter coefficient values to form the first set of FIR filter coefficient values.

26. A method as recited in claim 21, wherein the FIR filter simulation file is selected from a group comprising:

a MAX-PLUS2 vector file;

a MATLAB SIMULINK model;

a MATLAB TESTBENCH model;

a Verilog model; and

a VHDL model.

27. A method as recited in claim 21, further comprising:

optionally building a multi-rate FIR filter selected from a group comprising: an interpolating FIR filter and a decimating FIR filter.

28. A method as recited in claim 27, wherein the FIR filter is fitted in a programmable integrated circuit.

29. A method as recited in claim 28, wherein the programmable integrated circuit is a programmable logic device.

30. A method as recited in claim 21, wherein the estimating comprises:

determining a filter symmetry; and

determining a filter type.

31. A method as recited in claim 30, further comprising:

if the filter type is a parallel filter type,

finding a size of a parallel tap delay line;

dividing the second set of filter coefficients into a number of groups;

finding a size of a ROM LUT for a plurality of partial products;

and

finding a size of an adder tree for the plurality of partial products.

32. A method as recited in claim 31, further comprising:

if there are additional groups, then

finding a size of a ROM LUT for a plurality of partial products;

and

finding a size of an adder tree for the plurality of partial products.

33. A method as recited in claim 32, further comprising:

if there are no additional groups, then

finding a size of additions of all previous adder trees of the plurality of partial products for all groups; and

calculating a parallel filter resource estimate.

34. A method as recited in claim 30, further comprising:

if the filter type is a serial filter type,

finding a size of a serial tap delay line;

dividing the second set of filter coefficients into a number N of groups;

evaluating a number of logic elements required to implement the FIR filter;

finding a size of all adder trees for all groups;

finding a size of a scaling accumulator for a number N clock cycles; and

calculating a serial filter required resource estimate.

35. A method as recited in claim 33, wherein the parallel filter resource estimate is a number of logic cells required to implement the parallel filter.

36. A method as recited in claim 34, wherein the serial filter resource estimate is a number of logic cells or EABs required to implement the serial filter.

37. A method of building a decimating filter by a compiler using a plurality of domain polyphases wherein each of the plurality of polyphases is represented by a serial filter and wherein a single clock domain is used for each serial filter, comprising:

applying a first clock rule when an input data width is less than or equal to a decimation factor; and

applying a second clock rule when an input data width is greater than the decimation factor

38. A method as recited in claim 37, wherein the first clock rule comprises:

setting a clock rate to an input data rate;

setting an output data rate equal to the input data rate divided by the decimation factor; and

holding the input data for a number N clock cycles where N is equal to the decimation factor such that all polyphases are switched through at every clock cycle;

39. A method as recited in claim 37, wherein the second clock rule comprises:

setting the clock rate equal to a first speed multiplication factor (SMF1) multiplied by the input data rate, wherein the SMF1 is a smallest integer such that the SMF1 multiplied by the decimation factor is greater than or equal to the input data width;

setting the output data rate equal to the SMF1 multiplied by the input data rate divided by the decimation factor; and

holding the output data rate for a quantity L clock cycles, wherein the quantity L is equal to the SMF1 multiplied by the decimation factor, such that all polyphases are switched through at every quantity SMF1 clock cycles.

40. A method as recited in claim 37, wherein the decimating FIR filter includes a final adder, wherein the single clock domain is used for the final adder.

41. A method of building an interpolating FIR filter by a FIR compiler using a plurality of domain polyphases wherein each of the plurality of polyphases is

represented by a serial FIR filter and wherein a single clock domain is used for each serial FIR filter, comprising:

applying a first clock rule when an input data width is less than or equal to an interpolation factor; and

applying a second clock rule when an input data width is greater than the interpolation factor.

42. A method as recited in claim 41, wherein the first clock rule comprises:

setting a clock rate to an output data rate;

setting an input data rate equal to the output data rate divided by the interpolation factor; and

holding the input data for a quantity P clock cycles where the quantity P is equal to the interpolation factor such that all polyphases are switched through at every clock cycle.

43. A method as recited in claim 41, wherein the second clock rule comprises:

setting the clock rate equal to a second speed multiplication factor (SMF2) multiplied by the output data rate, wherein the SMF2 is a smallest integer such that the SMF2 multiplied by the interpolation factor is greater than or equal to the input data width;

setting the input data rate equal to the SMF2 multiplied by the output data rate divided by the interpolation factor; and

holding the input data rate for a quantity R clock cycles, wherein the quantity R is equal to the SMF2 multiplied by the interpolation factor, such that all polyphases are switched through at every quantity SMF clock cycles.

44. A method as recited in claim 41, wherein the interpolating FIR filter includes a state machine that controls a select bus coupled to a final multiplexer, wherein the single clock domain is used for the state machine.